REMARKS

Claims 1-21 are pending in the present application. Claims 1, 4-7, 9-11, 14-17, 19-21 are amended. Claims 3 and 13 are canceled.

Claims 1, 11, and 21 are amended to recite "identifying a set of hardware dump information elements comprises determining a dump mode and identifying a set of static arrays, and calculating an amount of memory to allocate for a dump list based on the dump mode and sizes of the set of static arrays." These features are supported at least on page 7, lines 16-19 of the current specification.

Claims 4, 5, 14, and 15 are amended to provide proper antecedent basis and to be consistent with amended claims 1 and 11. Claims 6 and 16 are amended to recite "building the dump list comprises iterating through the list of static arrays." These features are supported at least on page 8, lines 25-26 of the current specification.

Claims 7 and 17 are amended to provide proper antecedent basis. Claims 9 and 19 are amended to provide proper antecedent basis and to recite "collecting hardware dump data for each entry of the dump list from each component to be scanned." These features are supported at least one page 8, lines 27-30 of the current specification. Claims 10 and 20 are amended to provide proper antecedent basis.

No new matter is added as a result of the above amendments. Reconsideration of the claims in view of the above amendments and the following remarks is respectfully requested.

I. <u>35 U.S.C. § 103(a), Alleged Obviousness, Claims 1-21</u>

The Office Action rejects claims 1-21 under 35 U.S.C. § 103(a) as being unpatentable over Yamashita (U.S. Publication No. 2002/0010882 A1) in view of Vachon (U.S. Patent No. 6,681,348 B1). This rejection is respectfully traversed.

As to claims 1, 11 and 21, the Office Action states:

> As per claim 1, Yamashita discloses:
> A method of executing a hardware dump, comprising:
> identifying a set of hardware dump information elements to collect (column 2, lines 44-53);

> building the dump list in the allocated memory (column 6, lines 51-67).
>
> Yamashita does not explicitly disclose: calculating an amount of memory to allocate for a dump list based on the identified set of hardware dump information elements. Vachon discloses these elements at Page 2, ¶ 24-25. Yamashita is concerned with transmitting as little data as possible remotely so as to speed up the transfer of diagnostic data. Vachon makes use of block allocation for data sizes to facilitate the high speed buffering of data in a transmission system carrying diagnostic data away from a processor core and scan chain. Thus it would have been obvious to one of ordinary skill in the art at the time of invention to incorporate the memory allocation features of Vachon into the mini-dump system of Yamashita and in doing so create an even faster core diagnostic data transfer system.

Office Action dated June 15, 2004, pages 2-3.

Amended independent claim 1, which is representative of claims 11 and 21 with regard to similarly recited subject matter, recites:

> 1. A method of executing a hardware dump, comprising:
> identifying a set of hardware dump information elements to collect, <u>wherein identifying a set of hardware dump information elements comprises determining a dump mode and identifying a set of static arrays;</u>
> <u>calculating an amount of memory to allocate for a dump list based on the dump mode and sizes of the set of static arrays;</u>
> allocating the calculated amount of memory; and
> building the dump list in the allocated memory.
> (emphasis added)

Neither Yamashita nor Vachon teaches or suggests <u>identifying a set of hardware dump information elements comprises determining a dump mode and identifying a set of static arrays</u> or <u>calculating an amount of memory to allocate for a dump list based on the dump mode and sizes of the set of static arrays</u>.

As discussed in the Abstract, Vachon is directed to a system for generating a summary or mini-dump file from a system or application crash dump or core dump file without the need for referencing a large symbol table file. A stand alone extraction tool is provided in Vachon for extracting pertinent information from the crash dump or core dump file by utilizing information in the referencing portions. The referencing portions contain references to certain pertinent information including references conventionally not found in crash dump files. The stand alone extraction tool then generates a summary or mini dump file of the crash dump file utilizing the extracted pertinent information.

As discussed in the Abstract, Yamashita is directed to an integrated circuit device that sends trace data generated by a central processing unit (CPU) to a debug device without loss. Yamashita further teaches a method of controlling the operation of the integrated circuit device. A trace buffer is connected via a parallel bus to a predetermined output terminal of the CPU. A buffer monitoring circuit is connected to an input terminal of the trace buffer and to the predetermined control terminal of the CPU. The CPU executes various types of data processing requested by a program and outputs trace data indicating an execution history. The trace buffer temporarily stores the trace data that is output in parallel by the CPU. When a usage amount of the trace buffer exceeds a preset threshold, the buffer monitoring circuit sends an interrupt signal BRKINT to the CPU to suspend the data processing of the CPU and, when a preset period of time elapses, releases the suspension of data processing of the CPU.

The Office Action alleges that Vachon teaches identifying a set of hardware dump information elements to collect at column 2, lines 44-53, which reads as follows:

> The crash dump file may be a complete crash dump file of an application. A stand alone extraction tool is also provided for extracting pertinent information from the crash dump or core dump file by utilizing information in the referencing portion. The stand alone tool then generates a summary or mini dump file of the crash dump file (e.g., 64K in size). The stand alone tool can be compiled executable code written in a programming language such as C and/or C++. The summary or mini dump file can be esily communicated over a network or saved to a portable memory device for analysis by a system developer or the like.

In the above section, Vachon merely teaches identifying pertinent information from a crash dump or core dump file by using information in a referencing portion, such that a mini dump file may be generated using the extracted information. Vachon does not teach or suggest identifying a set of hardware dump information elements comprises <u>determining a dump mode and identifying a set of static arrays</u>. To the contrary, Vachon uses a referencing portion of the crash dump file to identify information to be extracted from the dump file. Vachon does not use a dump mode or a set of static arrays to identify the information. There is no mention of a set of static arrays in the reference. Therefore, Vachon fails to teach the features of claims 1, 11, and 21 of the present invention.

Yamashita also does not teach or suggest identifying a set of hardware dump information elements by <u>determining a dump mode and identifying a set of static arrays</u>. Yamashita teaches collecting trace data from a CPU and stores it in a trace buffer. Yamashita does not teach or suggest determining a dump mode or identifying a set of static arrays. Yamashita is only concerned with collecting the execution history of the CPU. Since Yamashita is not concerned with collecting dump data, Yamashita would not be interested in determining a dump mode. Yamashita also fails to teach or suggest a set of static arrays. Therefore, Yamashita does not and would not teach identifying a set of hardware dump information elements by determining a dump mode and identifying a set of static arrays.

In addition, the Office Action alleges that while Vachon does not explicitly teach <u>calculating an amount of memory to allocate for a dump list based on a dump mode and sizes of the set of static arrays</u>, Yamashita teaches these features on page 2, paragraph 24-25, which reads as follows:

> The rotate instruction block **32** is connected to the rotate circuit **22**, the signal generation block **33** is connected to the shift registers **23-26**, and the number-of-shift-registers monitor block **34** is connected to the control terminal on the CPU core **1**.
>
> The input control block **21** of the trace buffer **3** checks the amount of trace data that is received in parallel bus **2**. Then, <u>based on the amount of trace data checked by the input control block **21**, the buffer pointer counter **31** of the buffer monitoring circuit **4** calculates the number of shift registers **23-26** in the trace buffer **3** to be used for storing the trace data</u>. (emphasis added)

In the above section, Yamashita only teaches calculating the number of shift registers necessary in the trace buffer for storing trace data, based on the amount of trace data checked by input control block. Yamashita does not teach or suggest calculating an amount of memory to allocate for a dump list based on a dump mode and sizes of the set of static arrays. In paragraph 6, Yamashita teaches that the collected trace data is execution history data on the central processing unit (CPU). Thus, Yamashita is only concerned with calculating the number of registers necessary for collecting all of the

CPU execution history. Yamashita is not concerned with calculating the amount of memory necessary for a dump list of a hardware dump.

In addition, Yamashita calculates the number of registers necessary based on the amount of trace data collected. Yamashita does not calculate an amount of memory necessary for a dump list based on a determined dump mode and sizes of static arrays. Yamashita does not even mention a dump mode or sizes of a set of static arrays, because Yamashita is not concerned with a dump list for hardware dump. Yamashita is only concerned with collecting trace data from a CPU. Therefore, there is no need for Yamashita to determine a dump mode or sizes of a set of static arrays. Thus, Yamashita does not teach the features of claims 1, 11, and 21 of the present invention.

Furthermore, the Office Action alleges that Yamashita is concerned with transmitting as little as possible remotely so as to speed up the transfer of diagnostic data. Vachon is concerned with using block allocation of data sizes to facilitate high speed buffering of data in a transmission system carrying diagnostic data away from a processor core and scan chain. The Office Action then alleges that "it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the memory allocate features of Vachon into the mini-dump system of Yamashita and in doing so create an even faster core diagnostic data transfer system." Applicants respectfully disagree.

Neither Vachon nor Yamashita teaches or suggests calculating an amount of memory to allocate for a dump list based on a determined dump mode and sizes of static arrays. While Vachon teaches, at column 4, lines 58-65, allocating memory for a crash dump file based on whether the crash dump file is a complete crash dump file or a kernel memory dump file, Vachon does not teach or suggest anything about sizes of static arrays.

Yamashita, on the other hand, teaches calculating the number of shift registers necessary to store trace data in a trace buffer, not a dump list for hardware dump. In addition, Yamashita does not teach or suggest determining a dump mode, because Yamashita is not concerned with collecting hardware dump data. Yamashita also does not mention anything about sizes of a set of static arrays.

Thus, it would not have been obvious for a person of ordinary skill in the art to include Yamashita's calculation technique in Vachon's system, because Yamashita only calculates number of registers necessary for trace data, not a dump list for hardware dump. Vachon only teaches allocating memory for a crash dump file based on the type of the dump file, not based on the dump mode and sizes of static arrays.

Even if a person of ordinary skill in the art were somehow motivated to combine the teaching of Yamashita and Vachon, the resulting combination would still not be calculating an amount of memory necessary to allocate for a dump list based on the dump mode and the sizes of the set of static arrays. The resulting combination would rather be calculating the number of shift registers necessary for storing trace data based on the type of dump file.

In view of the above, Applicants respectfully submit that neither Yamashita nor Vachon, either alone or in combination, teaches or suggests the features of claims 1, 11, and 21. At least by virtue of their dependency on claims 1 and 11 respectively, neither Yamashita nor Vachon teaches or suggests the features of dependent claims 2-10 and 12-20. Accordingly, Applicants respectfully request withdrawal of the rejection of claims 1-21 under 35 U.S.C. § 103(a).

In addition, neither Yamashita nor Vachon, either alone or in combination, teaches or suggests the specific features as recited in dependent claims 2-10 and 11-20. For example, with regard to claim 4, which is representative of claim 14 with regard to similarly recited subject matter, neither Yamashita nor Vachon teaches or suggests that <u>the set of static arrays is a complete set of static arrays if the dump mode is a complete dump</u>. The Office Action alleges that Vachon teaches these features at column 6, line 57 to column 7, line 23, which reads as follows:

> The mini dump file **80** includes a second referencing portion **84** and a specific data structures portion **86**. In one aspect of the invention, the first and second referencing portion **82** and **84** have a size of about 8 kilobytes, while the specific data structure portion **86** has a size of about 56 kilobytes for a total of 64 kilobytes. The specific data structures portion **84** includes a current thread data structure **88**, a current process data structure **90**, a current stack data structure **92**, a loaded module list data structure **94** and a memory management information block **96**.
>
> The current thread data structure **88** refers to the state of the executing thread (e.g., Ethread) at the time of the fault. The current

> process data structure **90** refers to the state of the process (e.g., EProcess) of the executing thread at the time of the fault. Typically, the process data structure **90** includes information relating to a private virtual address space, which is a set of virtual memory addresses that the process can use, an executable program which defines initial code and data that is mapped into the process's virtual address space, a list of open handles to various system resources, such as semaphores, communication ports and files that are accessible to all threads in the process, security context information, a unique identifier and at least one thread of execution. The current stack data structure **92** includes the state of the stack of the kernel (e.g., processor registers of the kernel at the time of the kernel's last execution) at the time of the fault. The loaded module list data structure **94** refers to the state of the drivers loaded by the operating system at the time of the fault. The memory management information refers to the state of the memory management system at the time of the fault in addition to information relating to the mapping of virtual memory addresses to physical memory addresses.

In the above section, Vachon teaches that the mini dump file includes a specific data structure portion, which includes a current thread data structure, a current process data structure, a current stack data structure, a load module list data structure, and a memory management information block. However, Vachon fails to teach or suggest that any of these data structures equates to a complete set of static arrays.

As described on page 2, lines 2-5 of the current specification, a set of static arrays includes constants, each representing a trace array or scan ring, for example, to be dumped. Vachon does not teach or suggest such feature. There is no teaching or suggestion in Vachon as to what format the data structures should be implemented. Vachon merely mentions what information the data structures should include.

Yamashita also does not teach or suggest a complete set of static arrays if the dump mode is a complete dump. In paragraph 26, Yamashita teaches that based on the number of shift registers calculated, the signal generation block generates a shift/load signal for the shift registers and this signal causes the four serially-connected shift registers to sequentially store the serially-converted trace data. Thus, Yamashita uses a number of shift registers to store trace data. Yamashita also fails to teach or suggest a complete set of static arrays in the shift registers or anywhere else in the system. Therefore, neither Yamashita nor Vachon teaches or suggests the features of claims 4 and 14.

With regard to claim 5, which is representative of claim 15 with regard to similarly recited subject matter, neither Yamashita nor Vachon teaches or suggests that the set of static arrays is a subset of static arrays if the dump mode is an abbreviated dump. The Office Action alleges that Vachon teaches these features at column 7, line 51 to column 8, line 2, which reads as follows:

> **FIG. 5** illustrates one particular methodology for generating a summary file from a crash dump file generated by a system crash. At step **100**, a crash dump component residing in the operating system generates a crash dump file of the state of the physical memory present in the system at the time of the fault. The crash dump file includes a referencing portion that provides references to specific data structures in the crash dump file. At step **110**, the extraction tool opens the crash dump file and extracts the referencing portion of the crash dump file to determine the locations of specific data structures. The extraction tool then extracts the specific data structures based on the locations referenced in the referencing portion at step **120**. At step **130**, the extraction tool opens a new summary file of a specific predetermined size and inserts a copy of the referencing portion and the specific data structures into the new summary file. The extraction tool then provides the new summary file with a second referencing portion that includes references to the specific data structures in the new summary file at step **140**.

However, in the above section, Vachon merely teaches that the extraction tool generates a new summary file and inserts a copy of the referencing portion and specific data structures into the new summary file. There is no teaching or suggestion of identifying a format for the referencing portion and the specific data structures, let alone identifying a subset of static arrays, as recited in claim 5. In addition, Vachon fails to mention an abbreviated dump mode. Vachon merely teaches a method of generating a summary file from a crash dump file when the system crashes. There is no mention as to what the dump mode is when the mini dump file is generated.

Yamashita also does not teach or suggest these features. As described above in arguments presented in claim 4 and 14, Yamashita only teaches using a number of shift registers to store trace data, but fails to mention identifying any static array in the shift registers or anywhere else in the system. Furthermore, there is no teaching or suggestion of an abbreviated dump mode in Yamashita. All trace data outputted by the CPU is converted by the rotate circuit. Therefore, neither Yamashita nor Vachon teaches or suggests the features of claims 5 and 15.

With regard to claim 7, which is representative of claim 17 with regard to similarly recited subject matter, neither Yamashita nor Vachon teaches or suggests a set of static arrays that comprises a component static array for each component to be scanned. The Office Action alleges that Vachon teaches these features at column 9, lines 24-40, which reads as follows:

> **FIG. 9** illustrates a block diagram of components residing in an example of a user mode mini or summary crash dump file **280** generated by the extraction tool **240** illustrated in **FIG. 8**. The mini dump file **280** includes a first referencing portion **282** generally matching references of a referencing portion of the crash dump file that the user mode extraction tool **240** utilized to generate the user mode mini dump file **280**. The user mode mini dump file **280** includes a second referencing portion **284** and a specific data structure portion **286**. The specific data structure portion **286** includes a loaded module list data structure **288**, an exception, record data structure **290**, a first thread structure **292**, a second thread data structure **294** up to an nth thread data structure **296**. The loaded module list data structure **288** refers to the state of the driers loaded by the application program at the time of the fault. The exception record data structure **290** is a block of data that describes where a fault occurred.

In the above section, Vachon teaches a user mode mini dump file that includes first and second referencing portions and a number of specific data structure portions that are referenced by the second referencing portion. The data structure portions include a loaded module list data structure, which refers to the state of drivers loaded by application program at time of the fault; an exception, which describes where a fault occurs; and first thread up to nth thread structures, which refer to states of various threads at the time of the fault. However, none of these data structure portions comprises a component static array for each component to be scanned.

Each of the thread structures in Vachon is used to collect the state of a thread of execution in the application program. A thread of execution is different from a component to be scanned, such as a processor or a memory controller. Vachon specifically teaches that the data structure portions are used to collect information about a thread of execution, as opposed to a component, which is a hardware component. In addition, there is no teaching or suggestion in Vachon that each data structure portion includes a component static array. Vachon does not teach or suggest in what format the data structure portions should be implemented.

Yamashita also does not teach or suggest a set of static arrays that comprises a component static array for each component to be scanned. Rather, Yamashita uses a number of shift registers to store trace data that is outputted by only a single component, the CPU. Therefore, neither Yamashita nor Vachon teaches or suggests the features of claims 7 and 17.

As to dependent claim 8, which is representative of claim 18 with regard to similarly recited subject matter, neither Yamashita nor Vachon teaches or suggests that each component static array comprises a set of constants, each constant representing a hardware dump information element to be collected. The Office Action alleges that Vachon teaches these features at column 9, lines 24-40, which is reproduced above.

However, as discussed in arguments presented for claims 7 and 17, neither Yamashita nor Vachon teaches or suggests a component static array for each component to be scanned. Therefore, neither Yamashita nor Vachon would teach or suggest a component static array that comprises a constant for each hardware dump information element to be collected. Since Vachon only teaches collecting states of threads of execution of an application program and not hardware components, Vachon does not and would not teach or suggest a set of constants or that each constant represents a hardware dump information element to be collected.

Yamashita also does not teach or suggest a set of constants or each constant representing a hardware dump information element to be collected, because Yamashita only collects trace data for the CPU component and converts it from parallel to serial trace data, such that a debug device may analyze the trace data. Yamashita does not collect trace data from any other component. Therefore, neither Yamashita nor Vachon teaches or suggests the features of claims 8 and 18.

With regard to claim 9, which is representative of claim 19 with regard to similarly recited subject matter, neither Yamashita nor Vachon teaches or suggests <u>collecting hardware dump data for each entry of the dump list from each component to be scanned</u>. Vachon only teaches generating a mini dump file from referencing portions of the crash dump file or kernel memory dump file. Vachon does not collect dump data from each component to be scanned. Yamashita only teaches collecting execution history of only one component, the CPU, by storing trace data in shift registers of a trace buffer. Yamashita does not teach collect dump data for each entry of a dump list from each component to be scanned. Therefore, neither Yamashita nor Vachon teaches the features of claims 9 and 19.

With regard to claim 10, which is representative of claim 20 with regard to similarly recited subject matter, neither Yamashita nor Vachon teaches or suggests hardware dump information elements that comprise at least one of a scan ring, a trace array, cache contents, and cache directory contents. The Office Action alleges that Vachon teaches these features at column 4, lines 20-67, which reads as follows:

> The term "component" in the specification and the claims refers to a computer-related entity, either hardware, a combination of hardware and software, software, or software in execution. For example, a component may be one or more processes running on a processor, one or more objects, one or more executables, one or more threads of execution, one or more programs, a processor and a computer. By way of illustration, both an application running on a server and the server can be components.
>
> **FIG. 1** illustrates an eample of a machine **10** including a physical memory component **12** having a user mode portion **14** and a kernel mode portion **16**. The user mode portion **14** includes a plurality of processes **18** (e.g., application program processes), or portions of processes, residing in an upper portion or the user mode portion **14** of the physical memory **12**. The kernel mode portion **14** includes a number of processes (e.g., operation system processes), or portion of processes, residing in a lower portion or the kernel mode portion **16** of the physical memory **12**. The processes running in the kernel mode portion **16** include a memory manager process **20**, a kernel process **24**, a crash dump process **26**, one or more loaded modules **28** and other operating system processes **22**. Each of the above processes can include any number of threads associated with a corresponding process. The threads are entities within the process that the kernel schedules for execution in the processor of the machine.
>
> If the machine crashes (e.g., typically due to errors occurring in the kernel mode portion of physically memory), the kernel is interrupted and calls the crash dump process **26**, which creates a crash dump file **30** reflecting the state of the current physically memory **12** when the fault occurred. In the present invention, the crash dump process **26** is adapted to provide a referencing portion **32** in the crash dump file **30** that contains specific reference location information to specific data structures residing in a physical memory data structure portion **34**. The specific data structures provide information well suited for a developer in determining a cause of the fault. Typically, the crash dump file **30** is a complete crash dump file. Alternatively, the crash dump file **30** may be a kernel memory dump file. A complete memory dump file contains all of the physical memory present on the system at the time of the crash. This type of dump

requires that a file be at least the size of physical memory. A kernel memory dump contains only the kernel-mode read/write pages present in physical memory at the time of the crash. A kernel memory dump does not contain pages belonging to user processes.

However, nowhere in the above section, or in any other section, does Vachon teach or suggest hardware dump information elements, such as a scan ring, a trace array, cache contents or cache directory contents. The Examiner merely asserts that physical memory includes all the above components. However, Vachon specifically teaches, at column 4, lines 35-38, that the kernel mode portion includes a number of processes, such as operating system processes, or a portion of processes, residing in the lower portion or the kernel mode portion of the physical memory. Thus, only processes are included in the physical memory, not scan ring, trace array, cache contents, or cache directory contents.

Yamashita also does not teach or suggest hardware dump information elements as recited in claims 10 and 20. Yamashita only teaches a microprocessor that includes a trace buffer, which is connected to a debug device or tracer. There is no teaching or suggestion of hardware dump information elements, such as a scan ring, trace array, cache contents or cache directory contents, in the system of Yamashita. Therefore, neither Yamashita nor Vachon teaches or suggests the features of claims 10 and 20.

In addition to their dependency on claims 1 and 11, Applicants respectfully submit that neither Yamashita nor Vachon, either alone or in combination, teaches or suggests the specific features of claims 2-10 and 12-20. Accordingly, Applicants respectfully request the withdrawal of rejections to claims 2-10 and 12-20 under 35 U.S.C. § 103(a).

II. <u>Conclusion</u>

It is respectfully urged that the subject application is patentable over Yamashita and Vachon and is now in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

Respectfully submitted,

DATE: 10/13/04

[illegible]

Cathrine Kinslow
Reg. No. 51,886
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 367-2001
Attorney for Applicants

CK/im